12-15-2020

An Efficient Deep-Learning-Based Detection and Classification System for Cyber-Attacks in IoT Communication Networks

Qasem Abu Al-Haija
Tennessee State University

Saleh Zein-Sabatto
Tennessee State University

Recommended Citation

Abu Al-Haija, Q.; Zein-Sabatto, S. An Efficient Deep-Learning-Based Detection and Classification System for Cyber-Attacks in IoT Communication Networks. Electronics 2020, 9, 2152. https://doi.org/10.3390/electronics9122152

electronics

MDPI

Article

An Efficient Deep-Learning-Based Detection and Classification System for Cyber-Attacks in IoT Communication Networks

Qasem Abu Al-Haija * and Saleh Zein-Sabatto

Department of Electrical and Computer Engineering, Tennessee State University, Nashville, TN 37209, USA; mzein@tnstate.edu

* Correspondence: Qabualha@Tnstate.edu

Received: 18 November 2020; Accepted: 7 December 2020; Published: 15 December 2020

Abstract: With the rapid expansion of intelligent resource-constrained devices and high-speed communication technologies, the Internet of Things (IoT) has earned wide recognition as the primary standard for low-power lossy networks (LLNs). Nevertheless, IoT infrastructures are vulnerable to cyber-attacks due to the constraints in computation, storage, and communication capacity of the endpoint devices. From one side, the majority of newly developed cyber-attacks are formed by slightly mutating formerly established cyber-attacks to produce a new attack that tends to be treated as normal traffic through the IoT network. From the other side, the influence of coupling the deep learning techniques with the cybersecurity field has become a recent inclination of many security applications due to their impressive performance. In this paper, we provide the comprehensive development of a new intelligent and autonomous deep-learning-based detection and classification system for cyber-attacks in IoT communication networks that leverage the power of convolutional neural networks, abbreviated as IoT-IDCS-CNN (IoT based Intrusion Detection and Classification System using Convolutional Neural Network). The proposed IoT-IDCS-CNN makes use of high-performance computing that employs the robust Compute Unified Device Architectures (CUDA) based Nvidia GPUs (Graphical Processing Units) and parallel processing that employs high-speed I9-core-based Intel CPUs. In particular, the proposed system is composed of three subsystems: a feature engineering subsystem, a feature learning subsystem, and a traffic classification subsystem. All subsystems were developed, verified, integrated, and validated in this research. To evaluate the developed system, we employed the Network Security Laboratory-Knowledge Discovery Databases (NSL-KDD) dataset, which includes all the key attacks in IoT computing. The simulation results demonstrated a greater than 99.3% and 98.2% cyber-attack classification accuracy for the binary-class classifier (normal vs. anomaly) and the multiclass classifier (five categories), respectively. The proposed system was validated using a K-fold cross-validation method and was evaluated using the confusion matrix parameters (i.e., true negative (TN), true positive (TP), false negative (FN), false positive (FP)), along with other classification performance metrics, including precision, recall, F1-score, and false alarm rate. The test and evaluation results of the IoT-IDCS-CNN system outperformed many recent machine-learning-based IDCS systems in the same area of study.

Keywords: deep learning; convolutional neural network; IoT networks; cyber-attack detection; classification

1. Introduction

The Internet of things (IoT) comprises a collection of heterogeneous resource-constrained objects interconnected via different network architectures, such as wireless sensor networks (WSNs) [1].

These objects or "things" are usually composed of sensors, actuators, and processors with the ability to communicate with each other to achieve common goals/applications through unique identifiers with respect to the Internet protocol (IP) [2,3]. Current IoT applications include smart buildings, telecommunications, medical and pharmaceutical, aerospace and aviation, environmental phenomenon monitoring, agriculture, and industrial and manufacturing processes. The basic IoT layered architecture is shown in Figure 1. It has three layers: the perception layer (consisting of edge devices that interact with the environment to identify certain physical factors or other smart objects in the environment), the network layer (consisting of a number of networking devices that discover and connect devices over the IoT network to transmit and receive the sensed data), and the application layer (consisting of various IoT applications/services that are responsible for data processing and storage). Most cyber-attacks target the application and network layers of the IoT system.

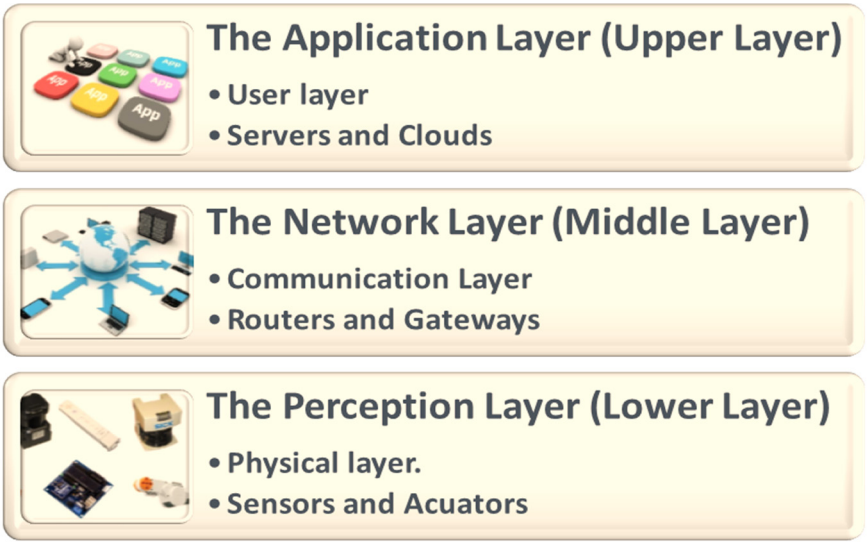

Figure 1. Internet of things (IoT) layered architecture considering the three-layer scheme of the IoT.

IoT is a promising profound technology with tremendous effects and potential for expansion. IoT infrastructures are vulnerable to cyber-attacks in that within the network, simple endpoint devices (e.g., thermostat, home appliance) are more constrained in computation, storage, and network capacity compared with the more complex endpoint devices (e.g., smartphones, laptops) that may reside within the IoT infrastructure. In fact, privacy, authentication, key management, trust management, and the cyber-attack identification are among the significant challenges of the IoT and cloud based IoT [4]. A number of studies were directed at addressing the security issues and challenges of IoT and cloud computing using a lightweight authentication process [5], and the secure data sharing and searching of the cloud based IoT [6]. Once the IoT infrastructure is breached, hackers have the ability to distribute the IoT data to unauthorized parties and can manipulate the accuracy and consistency of the IoT data over its entire life cycle [7]. Therefore, such cyber-attacks need to be addressed for safe IoT utilization. Consequently, vast efforts toward handling the security issues in the IoT model have been made in recent years. Many of the new cybersecurity technologies were developed by coupling the fields of machine learning with cybersecurity. It should be noted that the majority of new IoT attacks are slight deviations (i.e., mutations) of earlier known cyberattacks [8]. Such slight mutations of these IoT attacks have been demonstrated to be difficult to identify/classify using traditional machine learning techniques. Promising state-of-the-art research has been conducted for cybersecurity using deep neural networks [9–17]. Table 1 summarizes the research of conventional and traditional machine learning approaches to solve cybersecurity issues.

Table 1. Summary of the related research for machine-learning-based IoT security.

Research	Method	Description
G. Bendiab et al. 2020 [9]	Residual neural network (ResNet-50)	Two classes, utilized transfer learning with a 50-layer CovNet, employed pcap files containing pre-captured network traffic (normal/abnormal).
R. Shire et al. 2019 [10]	Convolutional neural network (CNN)	Five classes, employed CNN (single convolution layer + multilayer NN), three subsystems (sniffer-based traffic collection, American Standard Code For Information Interchange (ASCII)-based 2D traffic visualization, TensorFlow NN traffic analysis).
I. Baptista et al. 2019 [11]	Self-organizing incremental neural network (SOINN)	Five classes of malware with five filetypes, color-based binary visualization of ASCII for pre-captured files (.exe, .doc, .pdf, .txt, .htm).
K. Taher et al. 2019 [12]	Artificial neural network (ANN) with a support vector machine (SVM) classifier	Three classes, with 2 hidden layers and used only 35 features.
X. Gao et al. 2019 [13]	Deep neural network (DNN) with ensemble voting	Five classes with three methods: decision tree, random forest, K-nearest.
S. Sapre et al. 2019 [14]	Different machine-learning-based intrusion detection system (ML-IDS) techniques	Five classes, with two hidden layers and a naïve Bayes classifier.
S. Jan et al. 2019 [18]	ML-IDS-based SVM system	Only binary classification, used only two or three simple features.
M. Roopak et al. 2019 [19]	Deep neural network (DNN)	Small representative sample, did not reflect a realistic accuracy in actual IoT environments.
C. Ioannou et al. 2019 [20]	ML-IDS-based SVM system	Only used a binary classification, used an anonymous sensor topology.
O. Brun et al., 2018 [21]	Deep neural network (DNN)	System validation was poorly accomplished on a testbed comprising only three devices and naïve attacks were used to validate the system using real-time data with 50,000 samples.
V. Thing et al. 2017 [22]	Deep auto-encoder (DAE)	Unrealistic, very small dataset (no Distributed Denial of Service (DDoS), no probe), three hidden layers (256/128/64), needed significant time for feature engineering (FE).
P. Shukla et al. 2017 [23]	Neural network hybrid learning (K means plus decision trees)	Only used a binary classification, small-scale simulated network (16 nodes) with different topologies.
E, Hodo et al. 2016 [24]	Multi-layer perceptron (MLP) neural network	Unrealistic, small dataset with binary classes.
C. Kolias et al. 2016 [25]	Different ML-IDS techniques	Very time-consuming manual feature selection with four classes.
Y. Li et al. 2015 [26]	Hybrid NN (autoencoder + deep belief NN)	Redundant dataset needs to be up to date to reflect more rational results.

In this study, a new intelligent system that can detect the mutations of common IoT cyberattacks using non-traditional machine learning techniques exploiting the power of Nvidia-Quad GPUs was proposed. The proposed system employs the convolutional neural network (CNN), along with its associated machine learning algorithms to classify the NSL-KDD dataset records (we denote our system using the acronym IoT-IDCS-CNN). The NSL-KDD dataset stores non-redundant records of all the key

attacks of IoT computing with different levels of difficulties. Specifically, the main contributions of this paper can be summarized as follows (after Table 1):

- We provide a comprehensive efficient detection/classification model that can classify the IoT traffic records of an NSL-KDD dataset into two (binary classifier) or five (multiclassifier) classes. Furthermore, we present detailed preprocessing operations for the collected dataset records prior to its use with deep learning algorithms.
- We provide an illustrated description of our system modules and the machine learning algorithms. Furthermore, we demonstrate a comprehensive view of the computation process of our IoT-IDCS-CNN.
- We provide an inclusive development, validation environment, and configurations, along with extensive simulation results to gain insight into the proposed model and the solution approach. This includes simulation results related to the classification accuracy, classification time, and classification error rate for the system validation for both detection (binary classifier) and classification (multiclassifier).
- We provide a comprehensive performance analysis to gain more insight about the system efficiency, such as the confusion matrix to analyze the attacks' detection true/false positives and true/false negatives, along with other evaluation metrics, including precision, recall, the F-score metric, and the false alarm rate.
- We compare our findings with other related state-of-the-art works employing the same dataset, as well as with other state-of-the-art machine-learning-based intrusion detection systems (ML-IDSs) employing different datasets.

The rest of this paper is organized as follows: Section 2 introduces and justifies the dataset of IoT cyber-attacks employed by our system. Section 3 provides details of the proposed system architecture, development, and detailed design steps. Section 4 presents the simulation environment for system implementation, testing, and validation. Section 5 discusses the details about experimental evaluation, comparison, and discussion. Finally, Section 6 concludes the findings of the research.

2. Dataset of Cyber-Attacks

Data collection involves the gathering of information on variables of interest (VOSs) within a dataset in a documented organized manner that allows one to answer defined research inquiries, examine the stated hypotheses, and assess the output consequences. In this research, the variables of interest are concerned with the intrusions/attacks on data records in IoT computing environments. Two global datasets of IoT attacks can be investigated, including the KDD'99 dataset and the NSL-KDD dataset. Indeed, KDD'99 has been developed by the Defense Advanced Research Projects Agency(DARPA) intrusion detection evaluation program to build a network IDS that is capable of differentiating between "bad" and "good" connections [27]. This dataset includes a standard list of data to be inspected, which contains a broad range of cyber-attacks that are modeled in a military communication platform. However, one of the most important issues of this dataset is the enormous number of redundant data samples in the training and testing datasets. Such redundancy affects the accuracy of the classifier, which will have a bias toward more frequent records [27].

Lately, the original KDD'99 dataset [28] has been re-investigated and updated to include more up-to-date and non-redundant attack records with different levels of difficulty through the newer, reduced version called NSL-KDD [29,30]. Figure 2 shows sample records of the original NSL-KDD training dataset in the .csv format but read by Notepad in .txt format (prior to any processing technique). In this research, the NSL-KDD dataset was employed for many reasons, as follows:

(a) It can be efficiently imported, read, preprocessed, encoded, and programmed to produce two- or multiclass classification for IoT cyber-attacks.

(b) It covers all key attacks of IoT computing, including denial of service (DoS) [31], probe (side-channel) [32], root to local (R2L) [33], and user to root (U2R) [33].

(c) It is obtainable as a .txt/.csv filetype consisting of a reasonable number of non-redundant records in the training and test sets. This improves the classification process by avoiding the bias toward more frequent records.

(d) It correlates to high-level IoT traffic structures and cyberattacks, and it can be customized, expanded, and regenerated [29].

```
0,tcp,ftp_data,SF,491,0,0,2,2,0.00,25,0.17,0.03,0.1
0,udp,other,SF,146,0,0,13,1,0.00,0,0.00,0.60,0.88,0
0,tcp,private,S0,0,0,0,123,6,1.00,26,0.10,0.05,0.00
0,tcp,http,SF,232,8153,0,5,5,0.20,55,1.00,0.00,0.03
0,tcp,http,SF,199,420,0,30,32,0.00,255,1.00,0.00,0.
0,tcp,private,REJ,0,0,0,121,19,0.05,19,0.07,0.07,0.
0,tcp,private,S0,0,0,0,166,9,1.00,9,0.04,0.05,0.00,
0,tcp,private,S0,0,0,0,117,16,1.00,15,0.06,0.07,0.0
0,tcp,remote_job,S0,0,0,0,270,23,1255,23,0.09,0.05,
```

Figure 2. Sample records of the NSL-KDD training dataset.

The NSL-KDD dataset has been thoroughly developed with high-level diverse interpretations of the training data, which covers normal and abnormal IoT network traffic data. The normal data samples represent the legitimate data packets processed by the IoT network. The abnormal data samples represent mutated data packets (i.e., attacks) that are achieved by slight mutations in the previously developed attacks, such as small changes in the network packet header configurations. The original dataset is available in two classification forms: a two-class traffic dataset with binary labels and a multiclass traffic dataset that includes attack-type labels and a difficulty level. In both cases, it comprises 148,517 samples, each with 43 attributes, such as duration, protocol, and service [34]. The statistics of the traffic distribution of the NSL-KDD dataset are summarized in Table 2.

Table 2. Statistics of the traffic distribution of the NSL-KDD dataset [28].

Data Groups	Two-Class Dataset		Multiclass Dataset				
	Normal	Attack	Normal	DoS	Probe	R2L	U2R
Training	67,343	58,630	67,343	45,927	11,656	995	52
Testing	9711	12,833	9711	7458	2754	2421	200
Total	77,054	71,463	77,054	53,385	14,410	3416	252

R2L: root to local, U2R: user to root.

3. System Modeling

In this research, the proposed system was partitioned into distinct subsystems, each of which was implemented with several modules. Specifically, the system was composed of three subsystems including feature engineering (FE), feature learning (FL), and detection and classification (DC), as illustrated in Figure 3.

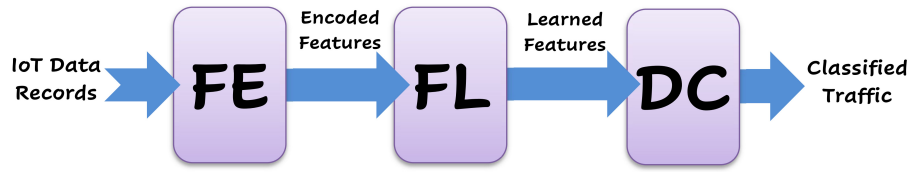

Figure 3. The three main subsystems comprising the proposed system. FE: feature engineering, FL: feature learning, DC: detection and classification.

3.1. Implementation of the Feature Engineering Subsystem

This subsystem is responsible for the conversion of raw IoT traffic data records of the NSL-KDD dataset into a matrix of labeled features that can be fed into and trained by the neural network's part of the FL subsystem. The implementation stages of this subsystem include the following.

(1). Importing the NSL-KDD dataset: In this stage, the collected dataset was imported/read using MATLAB 2019b (by MathWorks, Inc.) in a tabulated format instead of raw data in the original dataset text files. All data columns were assigned virtual names based on the nature of the data in the cells. The imported dataset includes 43 different features/columns. Figure 4 shows a sample of an imported NSL-KDD dataset using the table datatype. The illustrated sample shows only the first ten records, along with five features. All data columns were assigned virtual names based on the nature of data in the cells.

(2). Renaming categorical features: Four of the imported 43 features are categorical features that needed to be renamed prior to the data encoding and sample labeling processes. These features were the target protocol, the required service, the service flag, and the record category (e.g., normal or attack). Therefore, the four categorical columns were renamed accordingly in this stage. Figure 5 illustrates the four categorical features (columns) that were renamed for the binary-class data records (the other columns are omitted for better readability). Furthermore, note that the dataset encompasses multiclass data records for different traffic categories.

(3). One-hot encoding of categorical features: This module is responsible for the conversion of the categorical data records into numerical data records in order to be employed by the neural network. Therefore, three categorical features underwent a one-hot encoding process (1-N encoding) [35]. These features were the protocol column, the service column, and the flag column. The class feature/column was left for the sample-labeling process.

- For the protocol feature, three different types of protocols were revealed from the dataset, namely, TCP (Transmission Control Protocol), UDP (User Datagram Protocol), and ICMP (Internet Control Message Protocol). The one-hot encoding for this feature replaced the categorical data of the "protocol column" with the three numerical features, as shown in Table 3.
- For the service feature, 69 different services were revealed from the dataset, such as "AOL," "AUTH," "BGP," "COURIER," "CSNET_NS," ... , "UUCP_PATH," "VMNET," "WHOIS," "X11," and "Z39_50." The one-hot encoding for this feature replaced the categorical data of the "service column" with the 69 numerical features, as shown in Table 4.
- For the flags feature, 11 different flags were revealed from the dataset, namely, "OTH," "REJ," "RSTO," "RSTOS0," "RSTR," "S0," "S1," "S2," "S3," "SF," and "SH." The one-hot encoding for this feature replaced the categorical data of the "flag column" with the 11 numerical features, as shown in Table 5.

(4). Labeling the target feature: This stage is concerned with sample labeling using numerical (integer) labels for the target classes. Therefore, the categorical "class column" was converted to numerical classes according to the classification technique. In our system implementation, we considered two forms of traffic classifications: binary classification (1: normal vs. 2: attack) and multiple classifications (1: normal, 2: DoS, 3: probe, 4: R2L, and 5: U2R). After this stage, all data records were available in a numerical format (i.e., no categorical data existed anymore). As a result of 1-N encoding and numerical labeling, we converted the dataset into 123 features and one data label. The results of this stage, i.e., the encoded form of the dataset table of the two-class records, is provided in Figure 6.

(5). Converting tables to a double matrix: At the end of dataset importing, encoding, and labeling processes, the dataset samples and targets should be provided to the neural network inputs of FL subsystem as a matrix of all input numerical samples. Therefore, the encoded dataset tables

were converted to a double matrix (148,517 × 124). For instance, the following double matrix illustrates the first five rows of the dataset matrix.

$$\begin{bmatrix} [0 & 491 & 0 & \cdots & 0 & 1 & 0 & \cdots & 1 & \cdots & 0 & \cdots & 1 & 1] \\ 0 & 146 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 & \cdots & 0 & \cdots & 1 & 1] \\ 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 & 0] \\ 0 & 232 & 8153 & \cdots & 0 & 1 & 0 & \cdots & 0 & \cdots & 1 & \cdots & 1 & 1] \\ 0 & 199 & 240 & \cdots & 0 & 1 & 0 & \cdots & 0 & \cdots & 1 & \cdots & 1 & 1] \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix} \quad (1)$$

(6). Matrix resizing with a padding operation: This module is responsible for adjusting the size of the dataset matrix to accommodate the input size for the FL subsystem. This was performed by resizing the matrix of the engineered dataset from 148,517 × 124 to the new size of 148,517 × 784 since the input size of every individual sample processed at the FL subsystem was 28 × 28 =784. Thereafter, the new empty records of this matrix were padded with a zero-padding technique [36]. To avoid any feature biasing the samples of the dataset, the padded records were distributed equally around the data samples. Figure 7 illustrates an example of resizing with the zero-padding operation used in this research. The new matrix size was composed of 148,517 sample attacks, each with 784 features.

(7). Matrix normalization with a min-max norm: Data normalization is performed such that all the data points are in the same range (scale) with equal significance for each of them. Otherwise, one of the great value features might completely dominate the others in the dataset. Thus, this module is responsible for normalizing all integer numbers of the dataset matrix into a range between 0 and 1 using min-max normalization (MX-Norm) [37]. MX-Norm is a well-known method for normalizing data, as it is commonly used in machine learning applications. In this method, we scanned all the values in every feature, and then, the minimum value was converted into a 0 and the maximum value was converted into a 1, while the other values were converted (normalized) into a fractional value from 0 to 1. The min-max normalization X_i^{norm} for data record X_i at the ith position of matrix X is defined as follows:

$$X_i^{norm} = [X_i - \min(X)]/[\max(X) - min(X)]. \quad (2)$$

Furthermore, Figure 8 illustrates an example of the integer data features normalized using min-max normalization (0–1). The effect of normalization can be clearly seen as it ensured all features were at the same scale.

(8). Reshaping the double matrix: This module is responsible for creating the attack samples for the CovNet by reshaping the one-dimensional vectors of the attack records into two-dimensional square matrices to accommodate the input size for the developed CovNet network. Accordingly, every one-dimensional vector sample (1 × 784) was reshaped into a two-dimensional matrix (28 × 28) using a raw-by-raw reshaping fashion. This operation generated a square matrix for each data sample, as illustrated in Figure 9.

Table 3. Scheme for the replacement of the categorical data of the protocols.

Protocol	Equivalent One-Hot Encoding		
	TCP_Protocol	**UDP_Protocol**	**ICMP_Protocol**
TCP	1	0	0
UDP	0	1	0
ICMP	0	0	1

	Categorical	Categorical	Categorical	Number	Number
1	tcp	ftp_data	SF	491	0
2	udp	other	SF	146	0
3	tcp	private	S0	0	0
4	tcp	http	SF	232	8153
5	tcp	http	SF	199	420
6	tcp	private	REJ	0	0
7	tcp	private	S0	0	0
8	tcp	private	S0	0	0
9	tcp	remote_job	S0	0	0
10	tcp	private	S0	0	0

Figure 4. Imported NSL-KDD dataset: samples from the training dataset.

Table 4. Scheme for the replacement of the categorical data of the services.

Service	Equivalent One-Hot Encoding				
	AOL _Service	**AUTH_Service**	**BGP_Service**	**. . .**	**Z39_50_Service**
AOL	1	0	0	. . .	0
AUTH	0	1	0	. . .	0
BGP	0	0	1	. . .	0
.	.	.	.		.
.	.	.	.		.
.	.	.	.	⋱	.
Z39_50	0	0	0	. . .	1

	Protocol	Service	Flag	Class
1	tcp	ftp_data	SF	normal
2	udp	other	SF	normal
3	tcp	private	S0	attack
4	tcp	http	SF	normal
5	tcp	http	SF	normal
6	tcp	private	REJ	anomaly
7	tcp	private	S0	normal
8	tcp	private	S0	normal
9	tcp	remote_job	S0	anomaly
10	tcp	private	S0	anomaly

Figure 5. Imported NSL-KDD dataset: samples from the training dataset.

Table 5. Scheme for the replacement of the categorical data of flags.

Flag	Equivalent One-Hot Encoding				
	OTH_Flag	**REJ_Flag**	**RSTO_Flag**	**. . .**	**SF_Flag**
"OTH"	1	0	0	. . .	0
"REJ"	0	1	0	. . .	0
"RSTO"	0	0	1	. . .	0
.	.	.	.		.
.	.	.	.		.
.	.	.	.	⋱	.
"SH"	0	0	0	. . .	1

Features													Target Class
Time Dur.	Source bytes	Dest. bytes	...	TCP Protocol	UDP Protocol	ICMP Protocol	...	FTP Service	...	HTTP Service	...	SF Flag	
0	491	0		0	1	0		1		0		1	1
0	146	0		0	0	1		0		0		1	1
0	0	0		0	1	0		0		0		0	0
0	232	8153	...	0	1	0	...	0	...	1	...	1	1
0	199	420		0	1	0		0		1		1	1
⋮	⋮	⋮		⋮	⋮	⋮		⋮		⋮		⋮	⋮

Figure 6. Encoded dataset with labeling: sample from the training set. FTP: file transfer protocol, HTTP: hypertext transfer protocol.

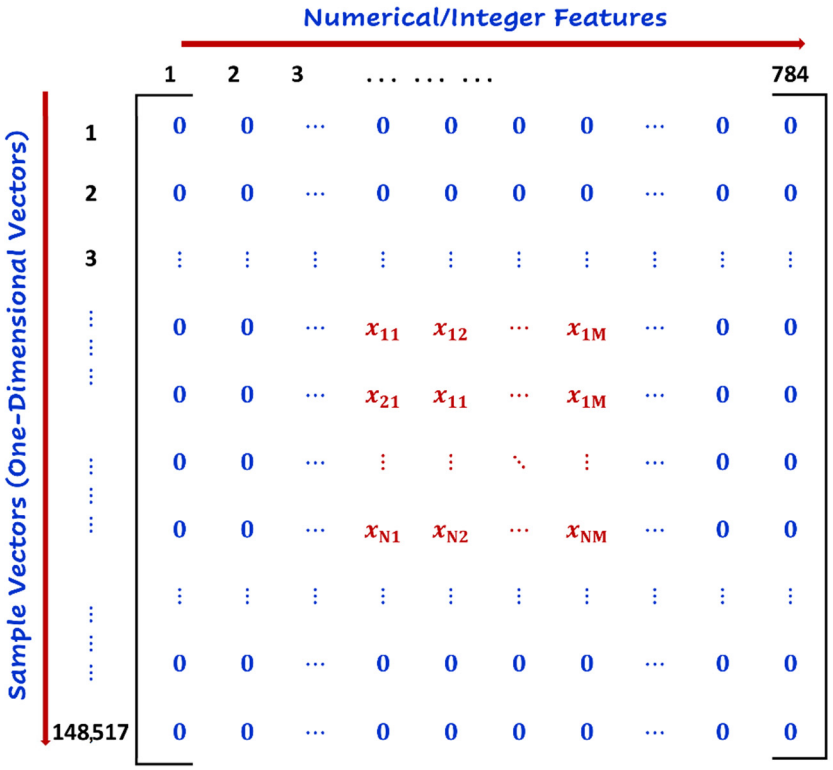

Figure 7. Encoded dataset with labeling: sample from the training set.

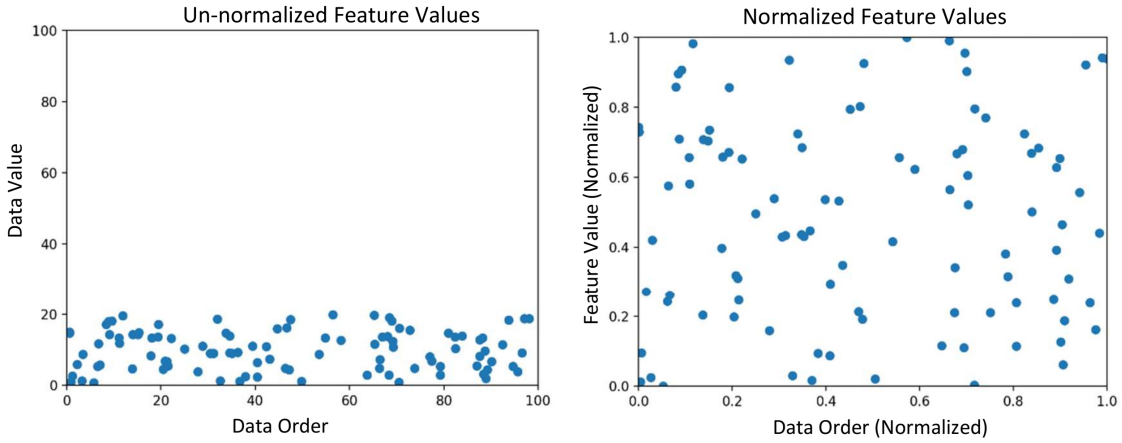

Figure 8. Illustration of the min-max normalization's impact on the data with different scales.

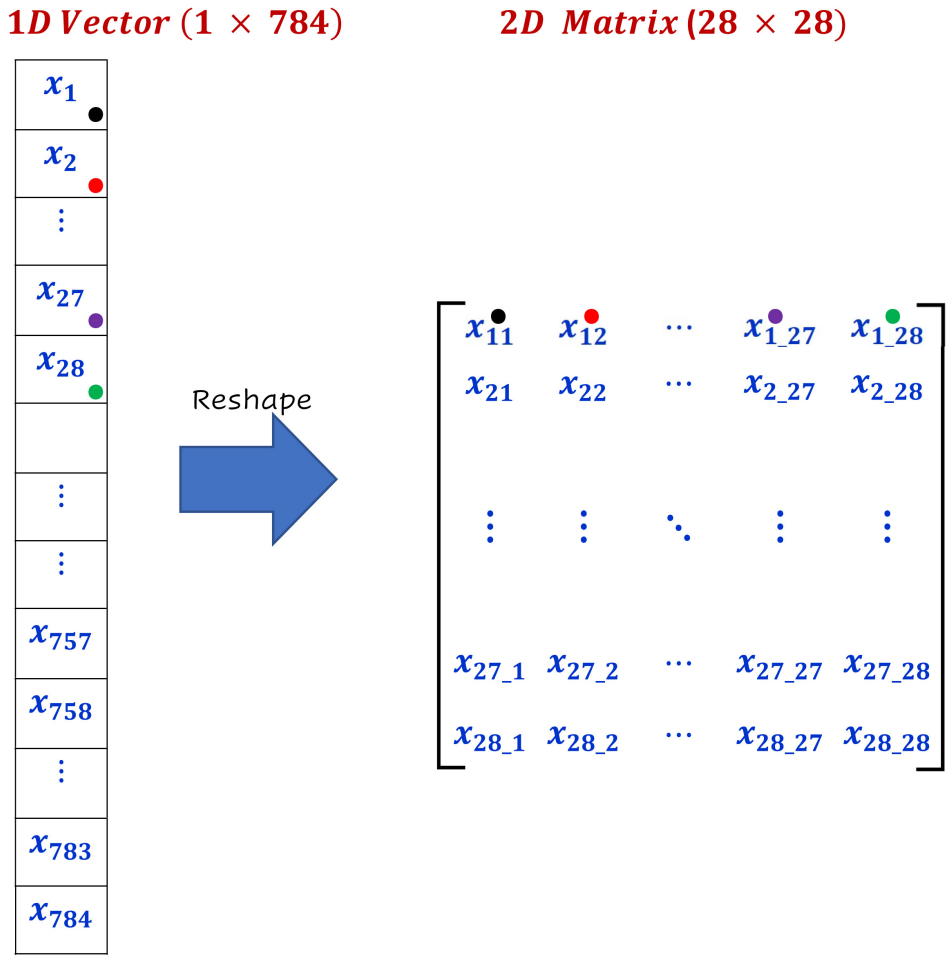

Figure 9. Illustration of the re-shaping operation of the dataset samples: the 1D vector was reshaped into a 2D matrix.

3.2. Implementation of Feature Learning Subsystem

So far, the development of the FE subsystem has been discussed and the next step was to process the encoded input features using an FL-subsystem-based CNN. The deep learning network was trained with a minimum classification error and thus a maximum accuracy. Generally, a CNN involves various layers, namely, convolution, activation, pooling, flatten, and others. Convolutional layers are the core

component of a CNN network and they are hierarchically assembled to generate a number of feature maps that enable CNNs to learn complex features, which are a vital operation for recognizing patterns in the classification and detection tasks. Therefore, the developed FL subsystem was responsible for an appropriate CNN that could accept the encoded features from the FE subsystem at the input layer and train on them with multiple hidden layers, as well as update the training parameters before classifying the IoT traffic dataset as being normal or an anomaly (mutated). The implementation stages of this subsystem included the following.

(1). Feature mapping with a 2D convolution operations layer: This module is responsible for generating new matrices called feature maps that emphasize the unique features of the original matrix [38]. These feature maps are produced by convolving (multiply and accumulate) the original matrix $n_{in} \times n_{in}$ using a number N of $k \times k$ convolution filters with padding size p and stride size s, which yields the feature maps $n_{out} \times n_{out}$. The size of the resultant feature maps can be evaluated as follows:

$$n_{out} = (n_{in} + 2p - k)/s + 1. \quad (3)$$

In this research, we applied 20 convolution filters (9×9) over the 28×28 input samples with $p = 0$ and $s = 1$, which resulted in 20 feature maps each (20×20). Figure 10 illustrates our convolutional layer, where the input was a 28×28 matrix and a filter of size 9×9, where this defined a space of 20×20 neurons in the first hidden layer. This was the case because we could only move the window 19 neurons to the right and 19 neurons to the bottom before hitting the right (or bottom) border of the input matrix. Note that the filter moves forward one position away, both horizontally and vertically, when a new row starts. Furthermore, note that the convolution layer goes through a backpropagation process to determine the most accurate values of its trainable parameters (weights: $k \times k \times N = 9 \times 9 \times 20$).

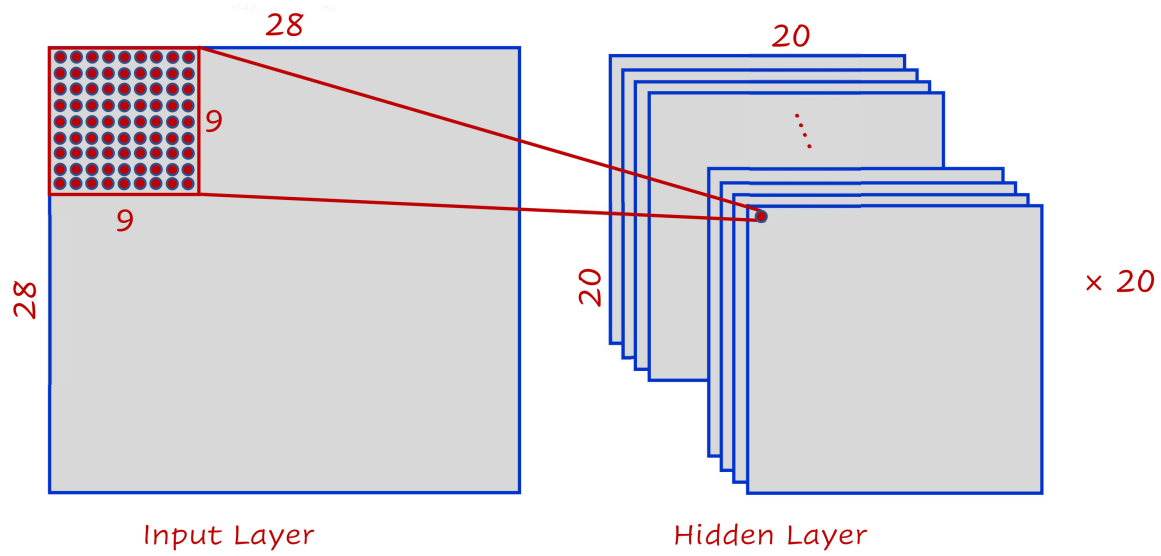

Figure 10. Implementation of convolution layer of our convolutional neural network (CNN).

(2). Feature activation with the Rectified Linear Unit (ReLU) function: This module is responsible for activating all units of the feature maps with a non-linear rectification function known as the ReLU. The ReLU function is MAX(X,0) that sets all negative values in the matrix X to zero, while all other values are kept constant. The reason for using ReLU is that training a deep network with ReLU tends to converge much more quickly and reliably than training a deep network with other non-linear activation functions, such sigmoid or tanh activation functions [39]. Figure 11 illustrates the rectification layer of the convolved maps.

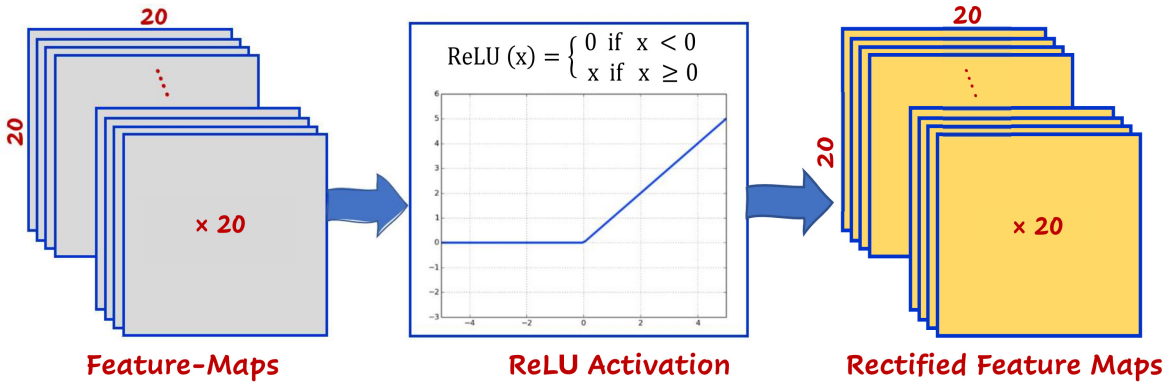

Figure 11. Implementation of ReLU activation layer of our CNN.

(3). Down-sampling with a pooling operations layer: This module is responsible for generating new matrices called pooled feature maps that reduce the spatial size of the rectified feature maps and thus reduce the number of parameters and computational complexity in the network [38]. This can be done by combining the neighboring points of a particular region of the matrix representation into a single value that represents the selected region. The adjacent points are typically selected from a fixed-size square matrix (determined according to the application). Among these points of the applied matrix, one value is nominated as the maximum or mean of the selected points. In this research, we used the mean pooling technique to develop the pooling layer since it combines the contribution of neighboring points instead of only selecting the maximum point. To produce the pooled-feature maps $L_{out} \times L_{out}$, the pooling filter $f \times f$ was independently applied over the rectified feature maps $L_{in} \times L_{in}$ with stride s, as follows:

$$L_{out} = (L_{in} - f)/s + 1 \quad (4)$$

In this research, we applied 20 pooling operations (2×2) over the 20×20 rectified feature maps with $s = 2$, which resulted in 20 feature maps each (10×10). Figure 12 illustrates our pooling layer, where the input from the previous layer was $20 \times 20 \times 20$ and the mean pooling filter was of size 2×2. Note that the stride value was 2, which means that the filter moves forward two positions away, both horizontally and vertically, when a new row starts. Thus, we ended up with pooled maps of size $10 \times 10 \times 20$.

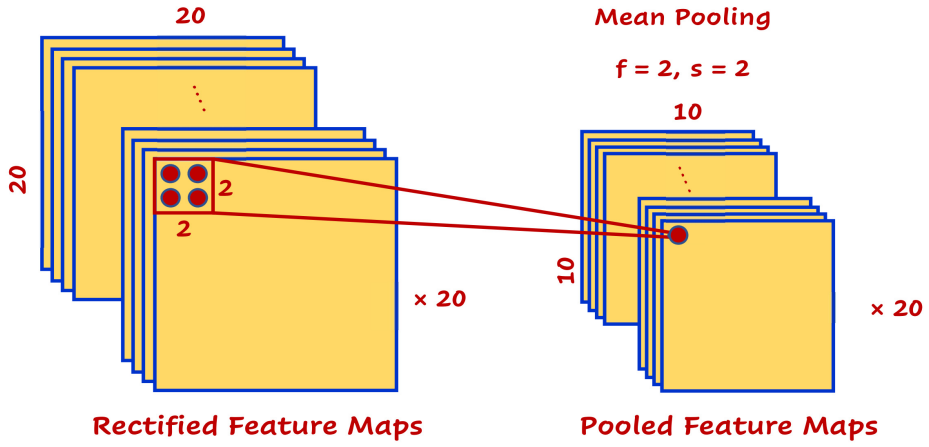

Figure 12. Implementation of pooling layer of our CNN.

3.3. Implementation of Detection and Classification Subsystem

The DC subsystem is responsible for providing traffic classification for the input traffic data into the binary-class classification (two classes: normal vs. anomaly) or multiclass classification (five classes: normal, DoS, probe, R2L, U2R). This subsystem is composed of three consecutive stages, as follows.

(1). Flattening layer of the pooled feature maps: This module is responsible for linearizing the output dimension of the convolutional/pooling layers network to create a single, long feature vector [38]. This can be achieved by converting the 2D data of the N pooled feature maps into a 1D array (or vector) to be inputted to the next layer, which is connected to the final classification model, called a dense or fully connected layer. Since the flattening layer collapses the spatial dimensions of the input into the channel dimension (array), this means that if the input to the flattening layer is N feature maps each with a dimension of $F_{in} \times F_{in}$, then the flattened output F_{out} can be obtained via the linear multiplication of the input dimensions by the number of maps:

$$F_{out} = N \times F_{in} \times F_{in}. \tag{5}$$

In this research, since we had 20 pooled feature maps ($N = 20$), each with dimensions of 10×10 ($F_{in} = 10$), then our flatten layer comprised 2000 nodes. Figure 13 illustrates the flattening layer development of our CNN.

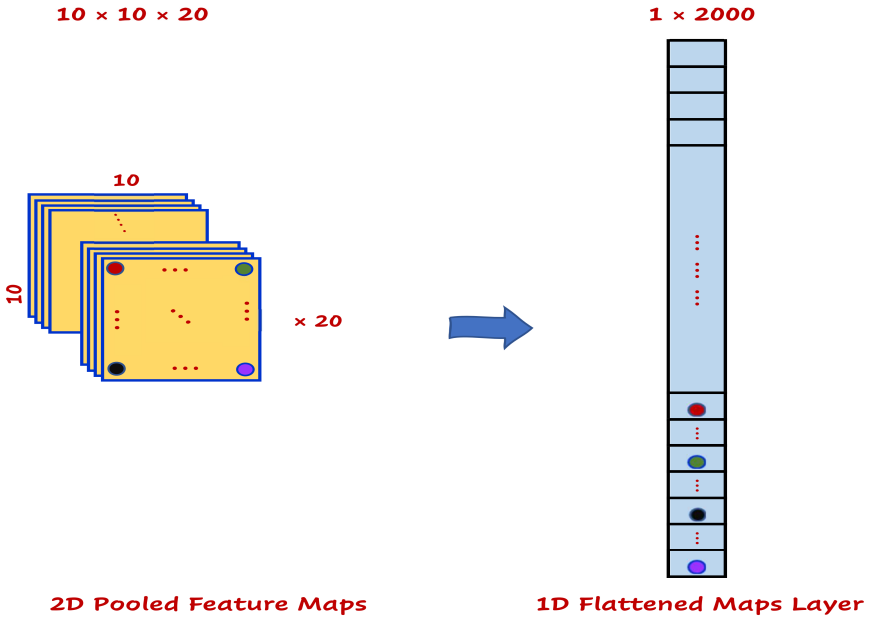

Figure 13. Implementation of the flattening layer (FTL) of our CNN.

(2). Fully connected (FC) layer with the ReLU function: FC layers, as the name implies, are those layers where all the inputs from one layer are connected to every activation unit of the next layer [38]. Commonly, FC layers are located as the last few layers of any CNN. Therefore, this module is responsible for compiling the high-level features extracted by previous layers (convolutional and pooling layers) into a reduced form of low-level features in which they can be used by the classifier located at the output layer to provide classification probabilities. In this research, we developed an FC layer with 200 neurons connected with 2000 nodes of the flattened (FLT) layer, which provided a layer complexity reduction of 10:1. As the inputs pass from the units of the FTL layer through the neurons of the FC layer, their values are multiplied by the weights and then pass into the employed activation function (normally the ReLU function), just in the same way as in a classical NN (i.e., a shallow NN). Thereafter, they are forwarded to the output classification layer, where each neuron expresses a class

label. Note that, the FC layer also goes through a backpropagation [38] process to determine the most accurate values of its trainable parameters (weights $W_{FTL} \times W_{FC} = 2000 \times 200$). Figure 14 illustrates the development of the FC layer of our CNN.

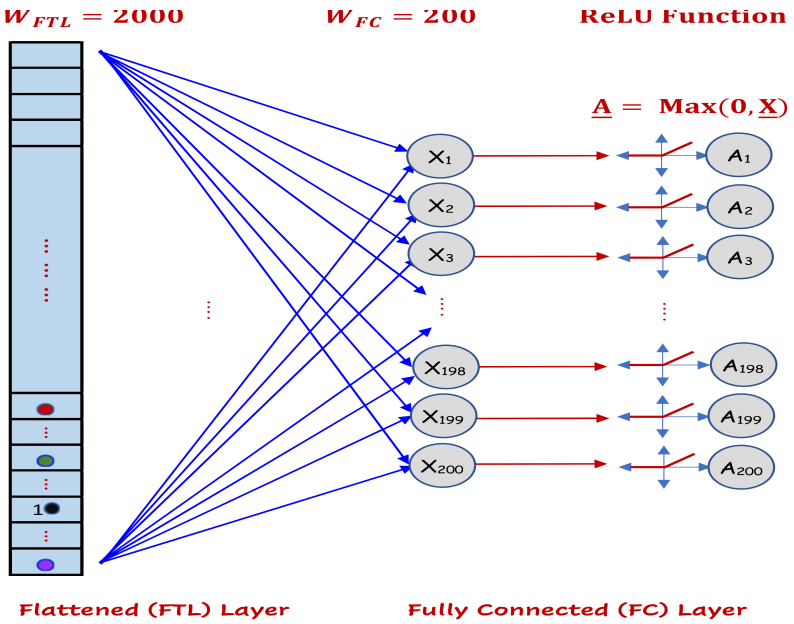

Figure 14. Implementation of the flattened layer of our CNN.

(3). Output layer with the softmax function: This module is responsible for providing/predicting the correct classification for each evaluated sample record of the utilized IoT attacks dataset. Here, we provided two types of classification, namely, the binary classifier (normal or anomaly) and the multiclassifier (normal, DoS, probe, R2L, U2R). The data points received from the 200 neurons of the FC layer ($A_1, A_2, \ldots, A_{200}$) were fully connected with the five neurons (C_1, C_2, C_3, C_4, C_5) of the output classes ($j = 5$ vectors) through the transposed weight connections (W_j^T). This is illustrated in Figure 15 and can be achieved algebraically as follows:

$$\underline{C} = \underline{W_j^T}.\underline{A} = \begin{bmatrix} \underline{W_1^T} \\ \underline{W_2^T} \\ \underline{W_3^T} \\ \underline{W_4^T} \\ \underline{W_5^T} \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ \vdots \\ \vdots \\ \vdots \\ A_{198} \\ A_{199} \\ A_{200} \end{bmatrix} = \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \end{bmatrix} \text{ where } \underline{W_1^T}, \underline{W_2^T}, \underline{W_3^T}, \underline{W_4^T}, \underline{W_5^T} \text{ are } 1 \times 200 \text{ vectors.} \quad (6)$$

Note that, the output layer also goes through a backpropagation process to determine the most accurate values of its trainable parameters (weights $W_{FC} \times W_{out} = 200 \times 5$). The last layer of the neural network is a softmax layer, which has the same number of nodes as the output layer. Softmax normalizes the output into a probability distribution on classes [38]. Specifically, softmax assigns numerical probability values for every class at the output layer, where these probabilities should sum up to 1.0 (following a probability distribution). Given an input a vector x of K real numbers, and i defines the index for the input values, then, the softmax function $\sigma: \mathbb{R}^k \longmapsto \mathbb{R}^k$ is defined as follows:

$$\sigma(x)_i = e^{x_i} / \sum_{j=1}^{K} e^{x_i} \text{ for } i = 1, 2, 3, \ldots, K \text{ and } x = (x_1, x_1, \ldots, x_K) \in \mathbb{R}^k. \quad (7)$$

For example, softmax might produce the following probabilities for an attack record shown in Table 6.

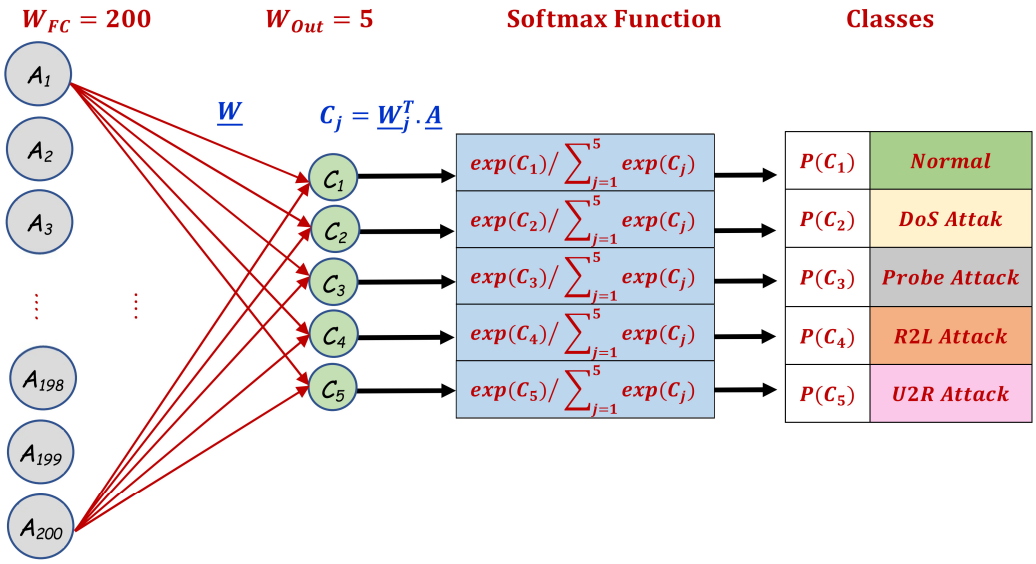

Figure 15. Implementation of the output layer using softmax in our CNN.

Table 6. Numerical Example of the outputs of Softmax function for five classes.

Item	Multiclass Dataset				
	Normal	DoS	Probe	R2L	U2R
Label	1	2	3	4	5
Probability	0.001	0.040	0.008	0.950	0.001

3.4. System Integration

In this section, we integrate all the aforementioned subsystems and modules to produce the complete system architecture of our IoT-IDCS-CNN. Figure 16 illustrates the top view architecture of the integrated system as a feedforward CovNet-network-based IoT attack detection system.

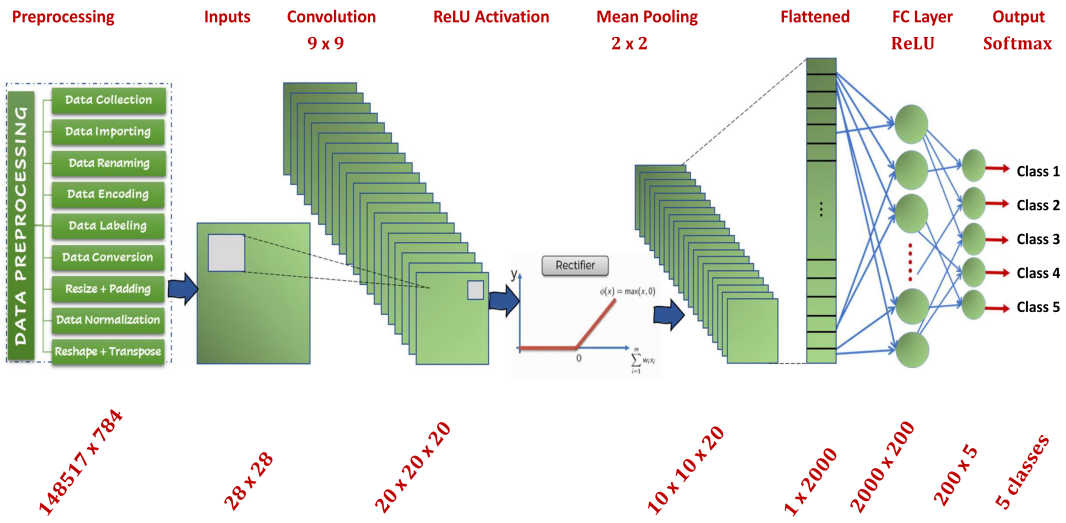

Figure 16. Top-view architecture of the proposed IoT-IDCS-CNN.

According to the system architecture, after the data preprocessing stages and using the 28 × 28 input matrix, we constructed 784 (= 28 × 28) input nodes. To extract the features of the input data, the network encompassed a deep convolutional layer involving a depth of 20 convolution filters of size 9 × 9. Thereafter, the results of the convolutional layer passed via the ReLU activation function, which was followed by the subsampling operation of the pooling layer. The pooling layer utilized the average pooling method with 2 × 2 submatrices. The pooled features were then flattened to 2000 nodes. The classification/detection neural network comprised the single hidden FC layer and the output classification layer. This FC layer comprised 200 nodes along with the ReLU activation function. Since our system required the classification of the data into five classes, the output layer was implemented with five nodes with the softmax activation function. The next table, Table 7, recaps the final integrated CovNet-based system for IoT attack detection.

Table 7. Summary of the developed CovNet for an IoT attack detection/classification system.

Layer	Comment	Trainable Parameters
Preprocessing	148,517 samples each (28 × 28)	-
Input	28 × 28 nodes (784 nodes)	-
Convolution	20 convolution filters (9 × 9) + ReLU	W_{Con} (9 × 9 × 20)
Pooling	Mean pooling (2 × 2)	-
Flattening	2000 nodes	-
Fully connected	200 nodes + ReLU	W_{FCL} (2000 × 200)
Output	5 nodes (or 2 nodes) + softmax	W_{Out} (200 × 5)

Moreover, the life cycle for the packet traffic received at the IoT gateway is provided in Figure 17 below. The input layer took the encoded features generated from the FE subsystem in order to be trained at the CNN, which updated the training parameters and generated the least cost/loss value (error) with optimal accuracy. The output layer employed the softmax classifier, which was used to classify the data using two classification techniques, namely, the binary classification technique, which provided two categories (normal vs. anomaly), and the multiclassification technique, which provided five categories (normal, DoS attack, probe attack, R2L attack, U2R attack).

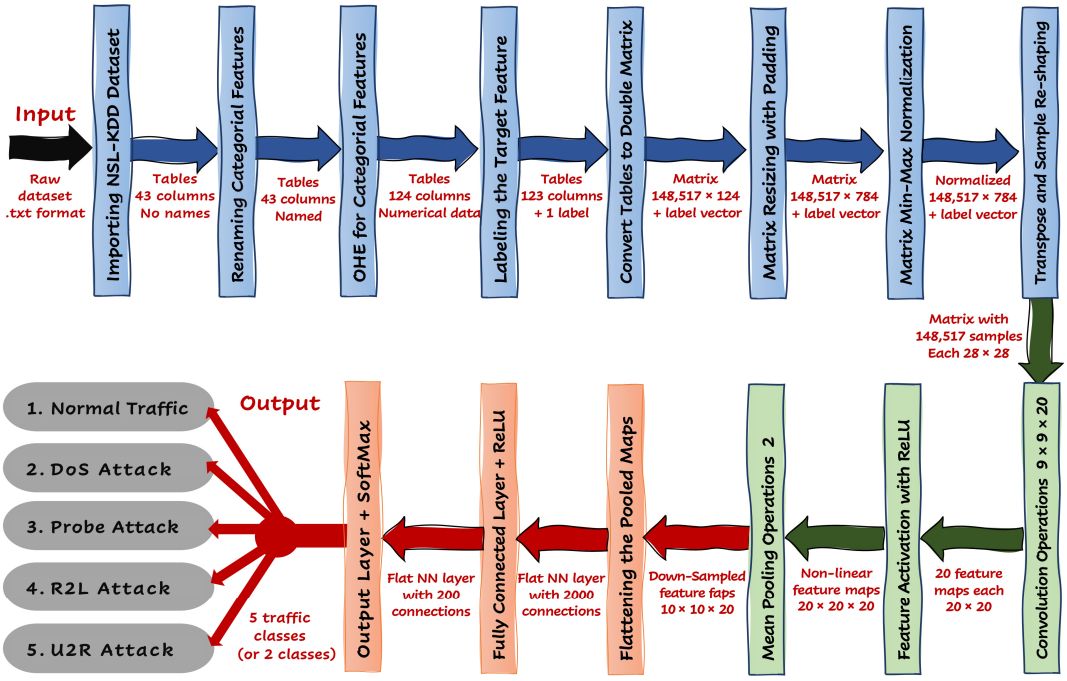

Figure 17. Comprehensive view of the computation process of the IoT-IDCS-CNN.

4. Simulation Environment

To implement, verify, and validate the proposed IoT attack detection and classification system, the training and testing were performed on the NSL-KDD dataset involving the key attacks against IoT communication. The classifier model was determined to have either two classes (binary attack detection) or five classes (multi-attack classification). The proposed system was implemented in MATLAB 2019a. To evaluate the system's performance, experiments were performed using a high-performance computing platform that utilized the power of a central processing unit (CPU) and a graphical processing unit (GPU) with the multicore structure of an NVIDIA GeForce® Quadro P2000 graphics card. The specifications for the workstation used in the development, validation, and verification are provided in Table 8.

Table 8. The system development and validation environment.

System Unit	Specifications
Processor Unit (CPU)	Intel Core I9-9900 CPU, 8 cores, @4900 MHz
Graphics Card (GPU)	NVIDIA Quad P2000@1480 MHz, 5 GB memory, 1024 CUDA cores
Cache Memory ($)	16 MB cache @ 3192 MHz
Main Memory (RAM)	32 GB DDR4 @ 2666 MHz
Operating System (OS)	64 bit, Windows 10 Pro
Hard Disk Drive (HD)	SATA 1TB drive + 256 GB SSD

Furthermore, the experimental setup for the training/testing model was configured as follows:

- Dataset Distribution:
  - 85% of the dataset was used for training (i.e., ≈128,500 data sample records).
  - 15% of the dataset was used for testing (i.e., ≈20,000 data sample records).
- CovNet Configurations:
  - Input (sample) size = 28 × 28.
  - Convolution kernel size = 9 × 9.
  - Activation function = ReLU.
  - Number of hidden layers = 5.
  - Number of kernels = 20.
  - Mean pooling filter size = 2 × 2.
  - Classifier function = softmax.
  - Number of output classes = 2 or 5.
- Model Optimization Configurations:
  - Optimization algorithm = mini batch gradient descent (find minimum loss).
  - Mini_batch_size(MBS) = 50, momentum factor (β) = 0.95, learning rate (α) = 0.05.
  - Momentum updates = $\mathrm{Mom_{Con}}[9 \times 9 \times 20]$, $\mathrm{Mom_{FCL}}[2000 \times 200]$, $\mathrm{Mom_{Out}}[200 \times 5]$.
  - All momentum updates were initialized using zeros matrices (zeros (size)).
- Training Model Configurations:
  - Training technique = backpropagation with momentum (to update weights).
  - Trainable weights = $\mathrm{W_{Con}}[9 \times 9 \times 20]$, $\mathrm{W_{FCL}}[2000 \times 200]$, $\mathrm{W_{Out}}[200 \times 5]$.
  - Backprop. Derivatives = $\mathrm{dW_{Con}}[9 \times 9 \times 20]$, d $\mathrm{W_{FCL}}[2000 \times 200]$, $\mathrm{dW_{Out}}[200 \times 5]$.
  - The number of epochs = 100 and the number of iterations per epoch ≈ 2500.
  - All trainable weights were initialized using a random number generator (rand).
  - All backpropagation derivatives were initialized using zeros matrices.

- Weight Update Policy:

— $dW_{Con} = dW_{Con}/MBS$, $dW_{FCL} = dW_{FCL}/MBS$, $dW_{Out} = dW_{Out}/MBS$
— $Mom_{Con} = \alpha \times dW_{Con} + \beta \times Mom_{Con}$; ➔ $W_{Con} = W_{Con} + Mom_{Con}$
— $Mom_{FCL} = \alpha \times dW_{FCL} + \beta \times$; ➔ $W_{FCL} = W_{FCL} + Mom_{FCL}$
— $Mom_{Out} = \alpha \times dW_{Out} + \beta \times Mom_{Out}$; ➔ $W_{Out} = W_{Out} + Mom_{Out}$

5. Results and Discussion

Verification and validation (V&V) are essential activities and quality control factors that are performed independently to check the system's compliance with requirements and specifications, and that it fulfills its intended purpose. Typically, the verification process is defined as a number of activities that are used to examine the suitability of the system or component (i.e., whether the product is being built right). On the other hand, the validation process is defined as a number of activities that are used to examine the conformity of the system (or any of its elements) with its purpose and functions (i.e., whether the right product is being built). Note that while system validation is distinct from verification, the actions of both processes are integral and are meant to be performed together [40]. In this section, we provide a comprehensive verification and validation to check the system's compliance with its intended objectives and purpose.

5.1. System Evaluation and Verification

To verify the effectiveness of the proposed system and whether it is in compliance with its intended functionalities and missions, we evaluated the system's performance using the recommended testing dataset in terms of the classification accuracy, classification error percent, and classification time, as follows:

$$\text{Classification Accuracy } (\%) = \frac{\text{Correctly Predicted Samples}}{\text{Number of Testing Samples}} \times 100\%, \tag{8}$$

$$\text{Classification Error } (\%) = \frac{\text{Incorrectly Predicted Samples}}{\text{Number of Testing Samples}} \times 100\%, \tag{9}$$

$$\text{Classification Time } (\text{ms}) = \sum_{i=1}^{\text{No. Runs}} \text{Execution time } (i) \times \frac{1000}{\text{No. Runs}}. \tag{10}$$

The plot for the overall testing classification accuracy and overall classification loss (classification error) that compares the performance of the binary classifier (two classes) and the multiclassifier (five classes) obtained during the validation process of the NSL-KDD dataset are illustrated in Figure 18. According to the figure, at the beginning and after one complete pass (epoch) of the testing process, both classifiers showed relatively low classification accuracy proportions with 85% and 79% registered for the two-class classifier and five-class classifier, respectively. Thereafter, both classification accuracy curves began to roughly increase with a stable tendency, while the testing epochs proceeded with faster and higher ceiling levels obtained for the classification accuracy of the two-class classifier. After training the system for 100 epochs, the system was able to record overall testing accuracy proportions of 99.3% and 98.2% for the two-class classifier and five-class classifier, respectively, for the given test dataset samples. Conversely, it can be clearly seen that both classifiers showed relatively high classification error proportions at the beginning of the testing process, with 15% and 21% registered for the two-class classifier and five-class classifier after one testing epoch, respectively. Thereafter, both classification error rates started to systematically decline, while the binary classifier progressed faster, achieving a 0.7% incorrect prediction proportion (classification error percentage). However, the classification error rate proportion for the multiclassifier was saturated with less than 2.0% of incorrect predictions. This range of classification error of both classifiers (0.7–1.8%) was permitted

to avoid underfitting or overfitting from the training loss (≈0.0%) and training accuracy (≈100%), and thus provided high-accuracy classification performance.

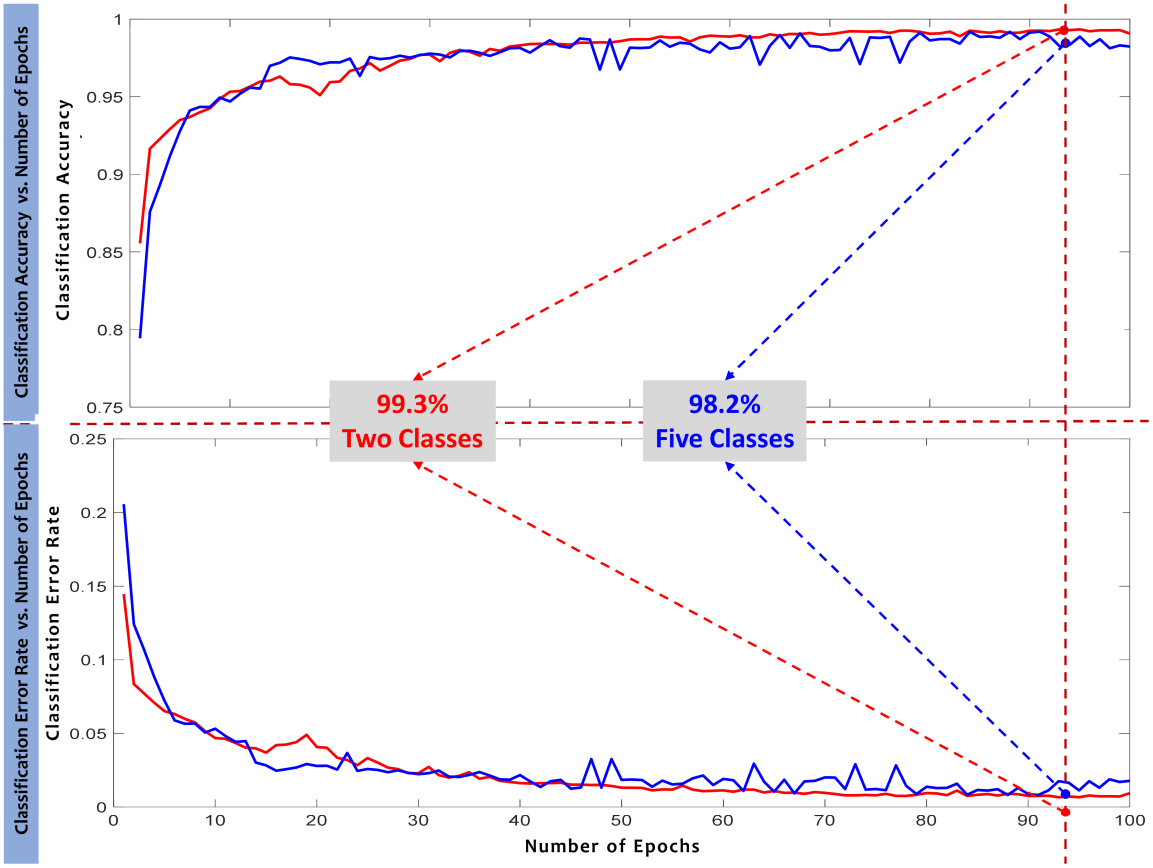

Figure 18. Testing detection/classification accuracy/error rate vs. the number of epochs.

Moreover, we analyzed the time required to perform the attack detection or classification for one IoT traffic sample. To obtain accurate and precise results, we ran the validation test 500 times and then computed the time statistics for the detection and classification. Figure 19 shows the detection/classification time performance for the proposed model (either two-class or five-class classifier). According to the figure, the time required to detect/classify one sample record ranged from min = 0.5662 ms to max = 2.099 ms with an average time of mean = 0.9439 ms recorded for the 500 simulation runs. This average time (around 1 ms) is very useful for the system to run in a dynamic environment, such as for real-time IDS applications.

Furthermore, even though the classification accuracy measurement is the key factor used to evaluate the efficiency of the classification or detection system, we evaluated the validation (testing) dataset using a confusion matrix [41] with a clear identification of the true positive (TP), true negative (TN), false positive (FP), and false negative (FN) analysis to provide more insight about the performance of the proposed system. Figure 20 shows the general confusion matrix of our system, confusion matrix results for the two-class classifier using the testing dataset and the confusion matrix results for the five-class classifier using the testing dataset.

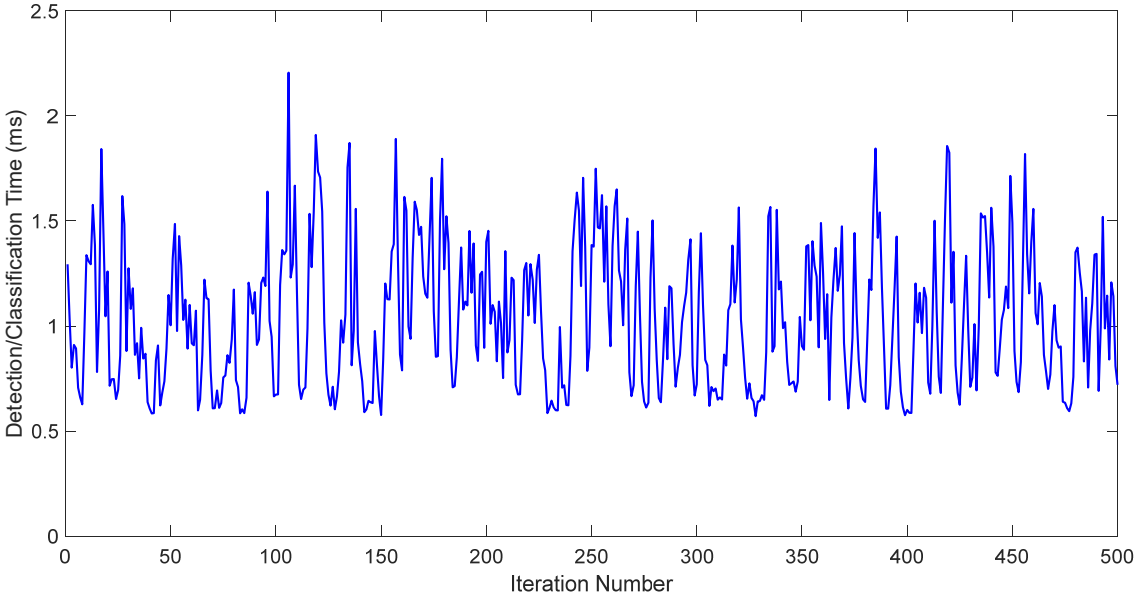

Figure 19. Run time performance of the IoT traffic classification over 500 simulation runs.

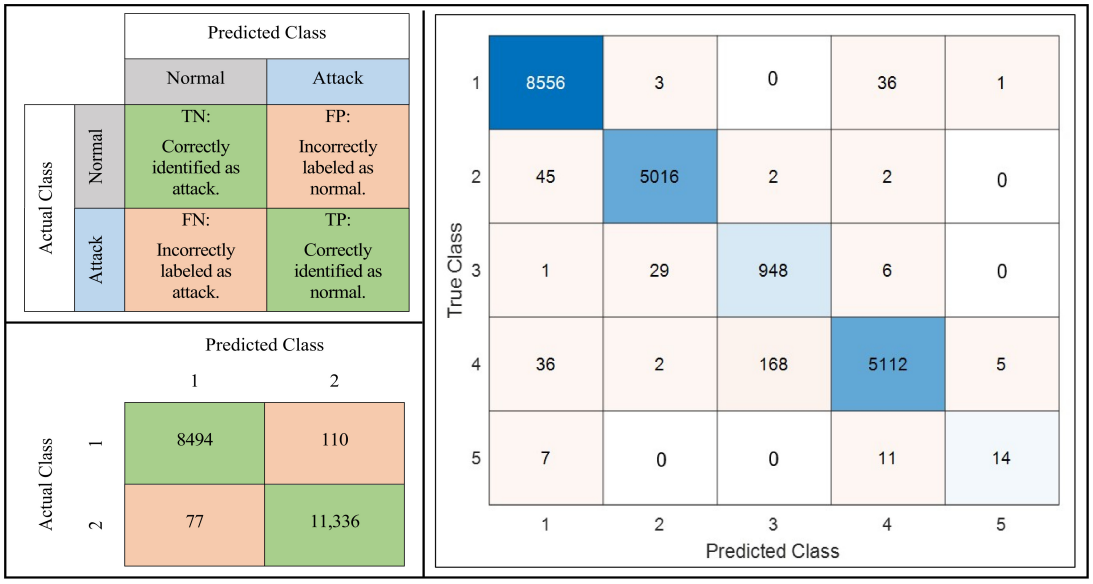

Figure 20. Confusion matrix analysis for both classification models: (**left**) two-class and (**right**) five-class.

The confusion matrix parameters (i.e., TN, TP, FN, FP) were used to compute some other performance evaluation metrics (which have less importance than the accuracy metric), namely, (a) the classification precision (detection rate), which is defined as the percentage of relevant instances (e.g., attacks) among the retrieved instances; (b) the classification recall (sensitivity), which is defined as the percentage of positive instances that are correctly labeled; (c) the F1-score, which is defined as the average score involving the precision and recall (i.e., utilizes both false negatives and false positives); (d) the false alarm rate, which is defined as the percentage of misclassified normal instances detected by the system [42]. These metrics can be calculated using the following equations, while Table 9 summarizes the results of the overall evaluation metrics for our proposed system:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \times 100\%, \tag{11}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \times 100\%, \tag{12}$$

$$\text{F} = 2 \times \frac{\text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}} \times 100\%, \tag{13}$$

$$\text{False Alarm Rate} = \frac{\text{FP}}{\text{TN} + \text{FP}} \times 100\%. \tag{14}$$

Table 9. Summary of the overall evaluation metrics results.

Evaluation Metrics	Two-Class Classification	Five-Class Classification
Correctly predicted samples	19860	19640
Incorrectly predicted samples	140	360
Classification accuracy	99.3%	98.2%
Classification error rate	00.7%	01.8%
Classification precision	99.04%	98.27%
Classification recall	99.33%	98.23%
F-score metric	99.18%	98.22%
False alarm rate (FAR)	01.28%	1.73%
Average classification time	0.9246	0.9439

5.2. System Validation and Benchmarking

To validate the proficiency of the proposed system in compliance with the system's purpose and specifications and to ensure a high level of reliability of our system's validation stage, we conducted a five-fold cross-validation process [43] that encompassed five different experiments for each classification model (total of 10 experiments) with different sets for training (≈128,000 samples) and validation (20,000 samples) nominated for each experiment, as demonstrated in Figure 21, which shows the distribution of the dataset across the folds for each conducted experiment.

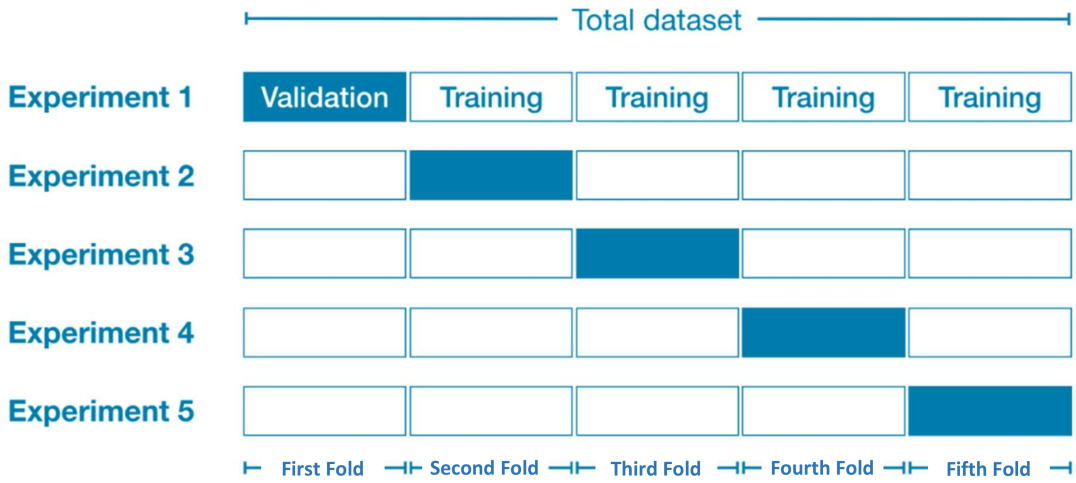

Figure 21. Scheme for the five-fold cross-validation of the proposed system.

For each experiment, we evaluated the validation accuracy and validation error for the classification system models (two classes/five classes). Thereafter, the results obtained from the five experiments were averaged to provide an overall validation accuracy and validation error values. Consequently, the proposed system provided a high level of stability and reliability across the dataset folds, which confirmed the system's robustness in the mission of attack detection and classification for IoT communications. The results of the five-fold cross-validation are provided in Table 10 below.

Table 10. The results of the five-fold cross-validation of both classifiers (accuracy and error).

Experiment	Two-Class		Five-Class	
	Accuracy	**Error**	**Accuracy**	**Error**
Experiment 1	0.9930	0.0070	0.9820	0.0180
Experiment 2	0.9942	0.0058	0.9950	0.0050
Experiment 3	0.98750	0.01250	0.9907	0.0093
Experiment 4	0.99440	0.00560	0.9929	0.0071
Experiment 5	0.99320	0.00680	0.9966	0.0034
Average	99.25%	0.75%	99.14%	0.86%

Additionally, to gain more insight into the advantages of the proposed method, we benchmarked the IoT-IDCS-CNN classification system by comparing its performance with other state-of-the-art machine-learning-based intrusion/attack detection systems in terms of the classification accuracy metric. For a better and more reasonable evaluation, we selected the related studies that employed machine learning techniques for intrusion/attack detection/classification for the NSL-KDD dataset (the same dataset used by our system) to be compared with our proposed IoT-IDCS-CNN. We summarize the classification accuracy metric values for the related state-of-the-art research in the following table, Table 11, in chronological order. Accordingly, it can be obviously noticed that the proposed IoT-IDCS-CNN model has a higher cyber-attack classification accuracy compared with other ML-IDS models by an improvement factor (IF) of ≈1.03–1.25.

Table 11. Comparison with state-of-the-art machine-learning-based intrusion detection systems (ML-IDSs) employing the same dataset (NSL-KDD).

Research	Accuracy	IF %
K. Taher et al. 2019 [12]	≈83.7%	117.3%
X. Gao et al. 2019 [13]	≈85.2%	115.2%
S. Sapre et al. 2019 [14]	≈78.5%	125.1%
M. Chowdhry et al. 2017 [15]	≈94.6%	103.8%
Q. Niyaz et al. 2016 [16]	≈88.4%	112.3%
I. Yadigar, et al. 2016 [17]	≈91.7%	108.0%
Proposed Method	≈98.2–99.3%	____

IF: improvement factor.

Finally, although the other existing related studies for machine-learning-based intrusion/attack detection/classification used different cyber-attack datasets, learning policies, programming techniques, and computing platforms, we can still compare the classification system performance in terms of testing accuracy metrics and the level of complexity for the developed method. Therefore, for better readability, we summarized the classification accuracy metrics for the other related state-of-art research in the following table, Table 12, in chronological order. According to the comparison done using the table, it can be seen that the proposed approach produced attractive results in terms of classification accuracy, showing superiority over all other compared methods.

Table 12. Comparison with state-of-the-art ML-IDSs employing different datasets.

Research	Data	Accuracy	IF %
G. Bendiab et al. 2020 [9]	Zero-Day Malware	≈94.50%	105.0%
R. Shire et al. 2019 [10]	Zero-Day Malware	≈91.32%	107.5%
I. Baptista et al. 2019 [11]	Ransomware filetypes	≈94.10%	104.4%
S.Jan et.al 2019 [18]	CICIDS Dataset	≈93.0%	106.7%
M. Roopak et al. 2019 [19]	CICIDS Dataset	≈92.0%	107.9%
C. Ioannou et al. 2019 [20]	Simulated Dataset	≈81.0%	122.5%
O. Brun et al., 2018 [21]	Real-Time Dataset	≈75.0%	132.4%

Table 12. *Cont.*

Research	Data	Accuracy	IF %
V. Thing et al. 2017 [22]	AWID Dataset	≈98.0%	101.3%
P. Shukla et al. 2017 [23]	Simulated Dataset	≈75.0%	132.4%
E. Hodo et al. 2016 [24]	DoS Dataset	≈99.0%	100.3%
C. Kolias et al. 2016 [25]	AWID Dataset	≈92.0%	107.9%
Y. Li et al. 2015 [26]	KDDCUP Dataset	≈92.0%	107.9%
Proposed Method	NSL-KDD Dataset	≈98.2–99.3%	____

6. Conclusions and Future Directions

An efficient and intelligent deep-learning-based detection and classification system for cyberattacks in IoT communication networks (called IoT-IDCS-CNN) was proposed, developed, tested, and validated in this study. The proposed IoT-IDCS-CNN makes use of high-performance computing by employing the robust Nvidia GPUs (Quad-Cores, CUDA-based) and the parallel processing employing the high-speed Intel CPUs (N-Cores, I9-based). For the purpose of the system development, the proposed IoT-IDCS-CNN was decomposed into three subsystems, namely, the feature engineering subsystem, the feature learning subsystem, and the detection and classification subsystem. All subsystems were individually developed and then integrated, verified, and validated in this research. Because of the use of a CNN-based design, the proposed system was able to detect and classify the slightly mutated cyberattacks of IoT networks (represented collectively by the NSL-KDD dataset, which includes all the key attacks found in IoT computing) with a detection accuracy of 99.3% between normal or anomaly traffic and could classify the IoT traffic into five categories with a classification accuracy of 98.2%. Furthermore, to ensure a high level of reliability for the system validation stage, we conducted a five-fold cross-validation process that encompassed five different experiments for each classification model. Moreover, to provide more insight about the performance of the system, the proposed system was evaluated using the confusion matrix parameters (i.e., TN, TP, FN, FP) and computed some other performance evaluation metrics, namely, the classification precision, the classification recall, the F1-score of the classification, and the false alarm rate. Finally, the experimental evaluation results of the IoT-IDCS-CNN system surpassed the results of many recent existing IDS systems in the same area of study. Several recommendations for future research works may be considered to extend this study. These further recommendations include the following:

(a) Additional data collection by setting up a real-time IoT communication network with a sufficient number of nodes and gateways to incorporate node diversity. A future researcher can develop a new software system that can catch and investigate any data packet communicated through the IoT environment (in-going and out-going) and come up with attacks to update an existing dataset or come up with a new dataset. Note that the packet collection and investigation should be performed for a sufficient amount of time to provide more insights into the type of packet (normal or anomaly) processed during IoT networking. This can provide different perceptions of the operation of the device, such as the utilization of the processing unit, the memory unit, and the communication traffic. The collected data can then be deemed as normal or an anomaly based on their behavior. For example, the normal data is related to the imitation of usual actions of local IoT devices, such as surveillance cameras. The anomaly data concerns botnet/probe actions, such as communication with command-and-control units. In the end, the data can be labeled accordingly.

(b) The proposed IoT-IDCS-CNN can be customized and used for intrusion detection by incorporating other cyberattack datasets, such as the Aegean Wireless Intrusion Dataset (AWID) dataset [44], the Canadian Institute for Cybersecurity-Intrusion Detection System (CICIDS) dataset [45], the Distributed Denial of Service (DDoS) dataset [46], and the University of New South Wales-New Bot 2015 (UNSW-NB15) dataset [47]. This can be achieved by customizing the preprocessing and

output layers accordingly with fine-tuning for the hidden layers, as well as the model parameters and hyperparameters to obtain the maximum classification accuracy and the lowest error rate.

(c) The proposed IoT-IDCS-CNN can also be tuned and used to perform other real-life applications that require image recognition and classification, such as medical, biomedical, and handwriting recognition applications.

(d) Finally, the proposed system can be employed by an IoT gateway device to provide intrusion detection services for a network of IoT devices, such as a network of Advanced RISC Machine (ARM) Cortex based nodes. More investigation on the proposed IoT-IDCS-CNN can be reported, including power consumption, memory utilization, communication, and computation complexity over low power IoT nodes with tiny system components (such as battery-operated/energy-aware devices).

Author Contributions: Conceptualization, Q.A.A.-H.; methodology, Q.A.A.-H.; software, Q.A.A.-H.; validation, Q.A.A.-H.; formal analysis, Q.A.A.-H.; investigation, Q.A.A.-H. and S.Z.-S.; resources, S.Z.-S.; data curation, Q.A.A.-H.; writing—original draft preparation, Q.A.A.-H.; writing—review and editing, S.Z.-S.; visualization, Q.A.A.-H.; supervision, S.Z.-S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded under the National Science Foundation Target Infusion Project (NSF-TIP) Program; titled "Targeted Infusion Project: Academic Enhancement of Electrical & Computer Engineering Program at Tennessee State University through IoT Research and Integrated Learning Environment" Award No. 1912313, funding period 2019–2022

Acknowledgments: Authors would like to thank the Department of Electrical and Computer Engineering in the College of Engineering at Tennessee State University for its administrative and technical support of this research.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Alrawais, A.; Alhothaily, A.; Hu, C.; Cheng, X. Fog Computing for the Internet of Things: Security and Privacy Issues. *IEEE Internet Comput.* **2017**, *21*, 34–42. [CrossRef]
2. Chiti, F.; Fantacci, R.; Loreti, M.; Pugliese, R. Context-aware wireless mobile automatic computing and communications: Research trends & emerging applications. *IEEE Wirel. Commun.* **2016**, *3*, 86–92. [CrossRef]
3. Silva, N.; Khan, M.; Han, K. Internet of Things: A Comprehensive Review of Enabling Technologies, Architecture, and Challenges. *IETE Tech. Rev.* **2017**, *35*, 1–16. [CrossRef]
4. Mahmoud, R.; Yousuf, T.; Aloul, F.; Zualkernan, I. Internet of things (IoT) security: Current status, challenges, and prospective measures. In Proceedings of the 10th International Conference for Internet Technology and Secured Transactions (ICITST), London, UK, 14–16 December 2015; pp. 336–341. [CrossRef]
5. Wazid, M.; Das, A.K.; BhBat, V.; Vasilakos, A.V. LAM-CIoT: Lightweight authentication mechanism in cloud-based IoT environment. *J. Netw. Comput. Appl.* **2020**, *150*. [CrossRef]
6. Mollah, M.B.; Azad, M.A.K.; Vasilakos, A. Secure Data Sharing and Searching at the Edge of Cloud-Assisted Internet of Things. *IEEE Cloud Comput.* **2017**, *4*, 34–42. [CrossRef]
7. Paar, J.P. *Understanding Cryptography*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 1–87. [CrossRef]
8. Caspi, G. Introducing Deep Learning: Boosting Cybersecurity with an Artificial Brain. Informa Tech, Dark Reading, Analytics. 2016. Available online: http://www.darkreading.com/analytics (accessed on 18 February 2019).
9. Bendiab, G.; Shiaeles, S.; Alruban, A.; Kolokotronis, N. IoT Malware Network Traffic Classification using Visual Representation and Deep Learning. In Proceedings of the 6th IEEE Conference on Network Softwarization (NetSoft), Ghent, Belgium, 29 June–3 July 2020; pp. 444–449. [CrossRef]
10. Shire, R.; Shiaeles, S.; Bendiab, K.; Ghita, B.; Kolokotronis, N. Malware Squid: A Novel IoT Malware Traffic Analysis Framework Using Convolutional Neural Network and Binary Visualization. In *Proceedings of the Internet of Things, Smart Spaces, and Next Generation Networks and Systems. NEW2AN 2019, ruSMART. Lecture Notes in Computer Science*; Springer: Cham, Switzerland, 2019; Volume 11660. [CrossRef]
11. Baptista, I.; Shiaeles, S.; Kolokotronis, N. A Novel Malware Detection System Based On Machine Learning and Binary Visualization. In Proceedings of the IEEE International Conference on Communications (IEEE ICC), Shanghai, China, 20–24 May 2019; pp. 1–6.

12. Taher, K.A.; Jisan, B.M.Y.; Rahman, M.M. Network Intrusion Detection using Supervised Machine Learning Technique with Feature Selection. In Proceedings of the International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST), Bangladesh, South Asia, 10–12 January 2019; pp. 643–646. [CrossRef]
13. Gao, X.; Shan, C.; Hu, C.; Niu, Z.; Liu, Z. An Adaptive Ensemble Machine Learning Model for Intrusion Detection. *IEEE Access* **2019**, *7*, 82512–82521. [CrossRef]
14. Sapre, S.; Ahmadi, P.; Islam, K. A Robust Comparison of the KDDCup99 and NSL-KDD IoT Network Intrusion Detection Datasets through Various Machine Learning Algorithms. *arXiv* **2019**, arXiv:1912.13204v1.
15. Chowdhury, M.M.U.; Hammond, F.; Konowicz, G.; Xin, C.; Wu, H.; Li, J. A few-shot deep learning approach for improved intrusion detection. In Proceedings of the 2017 IEEE 8th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON), New York, NY, USA, 19–21 October 2017; pp. 456–462. [CrossRef]
16. Javaid, A.; Niyaz, Q.; Sun, W.; Alam, M. A Deep Learning Approach for Network Intrusion Detection System. In Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS), New York, NY, USA, 24 May 2016; pp. 21–26. [CrossRef]
17. Imamverdiyev, Y.; Sukhostat, L. Anomaly detection in network traffic using extreme learning machine. In Proceedings of the 2016 IEEE 10th International Conference on Application of Information and Communication Technologies (AICT), Baku, Azerbaijan, 12–14 October 2016; pp. 1–4. [CrossRef]
18. Jan, S.U.; Ahmed, S.; Shakhov, V.; Koo, I. Toward a Lightweight Intrusion Detection System for the Internet of Things. *IEEE Access* **2019**, *7*, 42450–42471. [CrossRef]
19. Roopak, M.; Tian, G.Y.; Chambers, J. Deep Learning Models for Cyber Security in IoT Networks. In Proceedings of the 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 7–9 January 2019; pp. 0452–0457. [CrossRef]
20. Ioannou, C.; Vassiliou, V. Classifying Security Attacks in IoT Networks Using Supervised Learning. In Proceedings of the 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS), Santorini Island, Greece, 29–31 May 2019; pp. 652–658. [CrossRef]
21. Brun, O.; Yin, Y.; Gelenbe, E. Deep Learning with Dense Random Neural Network for Detecting Attacks against IoT-connected Home Environments. *Procedia Comput. Sci.* **2018**, *134*, 458–463. [CrossRef]
22. Thing, V.L.L. IEEE 802.11 Network Anomaly Detection and Attack Classification: A Deep Learning Approach. In Proceedings of the 2017 IEEE Wireless Communications and Networking Conference (WCNC), San Francisco, CA, USA, 19–22 March 2017; pp. 1–6. [CrossRef]
23. Shukla, P. ML-IDS: A machine learning approach to detect wormhole attacks in Internet of Things. In Proceedings of the 2017 Intelligent Systems Conference (IntelliSys), London, UK, 7–8 September 2017; pp. 234–240. [CrossRef]
24. Hodo, E.; Bellekens, X.; Hamilton, A.; Dubouilh, P.-L.; Iorkyase, E.; Tachtatzis, C.; Atkinson, R. Threat analysis of IoT networks using artificial neural network intrusion detection system. In Proceedings of the 2016 International Symposium on Networks, Computers and Communications (ISNCC), Yasmine Hammamet, Tunisia, 11–13 May 2016; pp. 1–6. [CrossRef]
25. Kolias, C.; Kambourakis, G.; Stavrou, A.; Gritzalis, S. Intrusion Detection in 802.11 Networks: Empirical Evaluation of Threats and a Public Dataset. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 184–208. [CrossRef]
26. Jang, S.-B.; Li, Y.; Ma, R.; Jiao, R. Collaborative Documentation Process Model. *Int. J. Softw. Eng. Its Appl.* **2015**, *9*, 219–232. [CrossRef]
27. Ozgur, A.; Erdem, H. A review of KDD99 dataset usage in intrusion detection and machine learning between 2010 and 2015. *PEERJ Prepr.* **2016**, *4*, e1954v1. [CrossRef]
28. Stolfo, S.; Fan, W.; Lee, W.; Prodromidis, A.; Chan, P. Cost-based modeling for fraud and intrusion detection: Results from the JAM project. In Proceedings of the DARPA Information Survivability Conference and Exposition. DISCEX'00, Hilton Head, SC, USA, 25–27 January 2000; Volume 2, pp. 130–144. [CrossRef]
29. Revathi, S.; Malathi, A. A Detailed Analysis on NSL-KDD Dataset Using Various Machine Learning Techniques for Intrusion Detection. *Int. J. Eng. Res. Technol. (IJERT)* **2013**, *2*, 1848–1853.
30. Canadian Institute for Cybersecurity (CIS). NSL-KDD Dataset. Available online: https://www.unb.ca/cic/datasets/nsl.html (accessed on 10 April 2019).
31. Kolias, C.; Kambourakis, G.; Stavrou, A.; Voas, J. DDoS in the IoT: Mirai and Other Botnets. *J. Computers* **2017**, *50*, 80–84. [CrossRef]

32. Ambedkar, C.; Babu, V.K. Detection of Probe Attacks Using Machine Learning Techniques. *Int. J. Res. Stud. Comput. Sci. Eng. (IJRSCSE)* **2015**, *2*, 25–29.
33. Pongle, P.; Chavan, G. A survey: Attacks on RPL and 6LoWPAN in IoT. In Proceedings of the 2015 International Conference on Pervasive Computing (ICPC), Pune, India, 8–10 January 2015; pp. 1–6. [CrossRef]
34. Bengio, Y.; Courville, A.; Vincent, P. Representation Learning: A Review and New Perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* **2013**, *35*, 1798–1828. [CrossRef]
35. Sarkar, D.J. Understanding Feature Engineering. Towards Data Science. Medium. 2018. Available online: https://towardsdatascience.com/tagged/tds-feature-engineering (accessed on 2 January 2019).
36. Brownlee, J. A Gentle Introduction to Padding and Stride for Convolutional Neural Networks. Deep Learning for Computer Vision, Machine Learning Mastery. 2019. Available online: https://machinelearningmastery.com/padding-and-stride-for-convolutional-neural-networks (accessed on 8 March 2019).
37. Kalay, A.F. Preprocessing for Neural Networks—Normalization Techniques. Machine Learning, Github.IO. Available online: https://alfurka.github.io/2018-11-10-preprocessing-for-nn.2018 (accessed on 14 May 2019).
38. Li, F. CS231n: Convolutional Neural Networks for Visual Recognition. Computer Science, Stanford University. 2019. Available online: http://cs231n.stanford.edu (accessed on 15 March 2019).
39. Brownlee, J. A Gentle Introduction to the Rectified Linear Unit (ReLU). Deep Learning for Computer Vision, Machine Learning Master. 2019. Available online: https://machinelearningmastery.com (accessed on 14 May 2019).
40. INCOSE. *INCOSE Systems Engineering Handbook, version 3.2.2*; INCOSE-TP-2003-002-03.2.2; International Council on Systems Engineering (INCOSE): San Diego, CA, USA, 2012.
41. Narkhede, S. Understanding Confusion Matrix. Medium: Towards data science. 2018. Available online: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62 (accessed on 12 January 2020).
42. Kim, P. *MATLAB Deep Learning with Machine Learning, Neural Networks and Artificial Intelligence*. Apress: Part of Springer Nature. 2017. Available online: https://www.apress.com/gp/book/9781484228449 (accessed on 10 October 2020).
43. Gupta, P. Cross-Validation in Machine Learning. Medium: Towards data science. 2017. Available online: https://towardsdatascience.com/cross-validation-in-machine-learning-72924a69872f (accessed on 13 February 2020).
44. Kolias, C.; Kambourakis, G.; Gritzalis, S. Attacks and countermeasures on 802.16: Analysis & assessment. *IEEE Commun. Surv. Tuts* **2013**, *15*, 487–514. [CrossRef]
45. CICIDS Dataset. DS-0917: Intrusion Detection Evaluation Dataset. Available online: https://www.impactcybertrust.org/dataset_view?idDataset=917 (accessed on 2 February 2020).
46. DDoS Dataset. Distributed Denial of Service (DDoS) attack Evaluation Dataset. Available online: https://www.unb.ca/cic/datasets/ddos-2019.html (accessed on 2 February 2020).
47. Moustafa, N.; Slay, J. UNSW-NB15: A comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In Proceedings of the 2015 Military Communications and Information Systems Conference (MilCIS), Canberra, ACT, Australia, 10–12 November 2015; pp. 1–6. [CrossRef]